Множественное наследование

Один класс может наследовать атрибуты двух и более классов одновременно. Для этого используется список базовых классов, в котором каждый из базовых классов отделен от других запятой. Общая форма множественного наследования имеет вид:

```
class имя_порожденного_класса: список базовых классов
{
...
};
```

В следующем примере класс Z наследует оба класса X и Y:

```
#include <iostream.h>
class X {
protected:
int a;
public:
void make_a(int i) { a = i; }
};
class Y {
protected:
int b;
public:
void make_b(int i) { b = i; }
};
// Z наследует как от X, так и от Y
class Z: public X, public Y {
public:
int make_ab() { return a*b; }
};
int main()
{
Z i;
i.make_a(10);
i.make_b(12);
cout << i .make_ab();
return 0;
}
```

Поскольку класс Z наследует оба класса X и Y, то он имеет доступ к публичным и защищенным членам обоих классов X и Y.

В предыдущем примере ни один из классов не содержал конструкторов. Однако ситуация становится более сложной, когда базовый класс содержит конструктор. Например, изменим предыдущий пример таким образом, чтобы классы X, Y и Z содержали конструкторы:

```
#include <iostream.h>
class X {
protected:
int a;
public:
X() {
a = 10;
cout << "Initializing X\n";
}
```

```
};
class Y {
protected:
int b;
public:
Y() {
cout << "Initializing Y\n";
b = 20;
}
};
// Z наследует как от X, так и от Y
class Z: public X, public Y {
public:
Z() { cout << "Initializing Z\n"; }
int make_ab() { return a*b; }
};
int main()
{
Z i;
cout << i.make_ab();
return 0;
}
```

Программа выдаст на экран следующий результат:

```
Initializing X
Initializing Y
Initializing Z
200
```

Обратим внимание, что конструкторы базовых классов вызываются в том порядке, в котором они указаны в списке при объявлении класса Z.

В общем случае, когда используется список базовых классов, их конструкторы вызываются слева направо. Деструкторы вызываются в обратном порядке — справа налево.

Передача параметров в базовый класс

До сих пор ни в одном из примеров наследования базовый класс не содержал конструкторов, использующих параметры. До тех пор, пока конструкторы не имеют аргументов, от производного класса не требуется каких-либо специальных действий. Однако, когда базовый класс имеет конструктор с аргументами, производные классы должны явным образом обрабатывать эту ситуацию путем передачи базовому классу необходимых аргументов. Для этого используется расширенная форма конструкторов производных классов, в которые передаются аргументы конструкторам базового класса. Эта расширенная форма показана ниже:

порожденный конструктор (список_аргументов): базовый 1(список_аргументов), базовый2(список_аргументов), ..., базовый3(список_аргументов)

```
{
...
}
```

Здесь под базовый1,... базовыйN обозначены имена базовых классов, наследуемые производным классом. Обратим внимание, что с помощью двоеточия конструктор производного класса отделяется от списка конструкторов базового класса. Список аргументов, ассоциированный с базовыми классами, может состоять из констант,

глобальных переменных или параметров конструктора производного класса. Поскольку инициализация объекта происходит во время выполнения программы, можно использовать в качестве аргумента любой идентификатор, определенный в области видимости класса.

Следующая программа иллюстрирует, как передаются аргументы базовому классу из производного класса:

```
   #include <iostream.h>
class X {
protected:
int a;
public:
X (int i) ( a = i; )
};
class Y {
protected:
int b;
public:
Y (int i) { b = i; }
};
// Z наследует как от X, так и от Y
class Z: public X, public Y {
public:
/* Инициализация X и Y через конструктор Z. Обратим внимание, что z на самом деле не использует x или y, но если потребуется, то может */
Z (int x, int y): X(x) , Y(y)
{
cout << "Initializing\n";
}
int make_ab() { return a*b; }
};
int main()
{
Z i (10, 20);
cout << i.make_ab();
return 0;
}
```

Обратим внимание, что конструктор класса Z фактически не использует параметры прямо. Вместо этого в данном примере они просто передаются конструкторам классов X и Y. Вместе с тем при необходимости, разумеется, класс Z может использовать эти и другие аргументы.